NASA/CR-2001-211235
ICASE Report No. 2001-32

An Analysis Mechanism for Automation in Terminal Area

Stavan M. Parikh
University of Virginia, Charlottesville, Virginia

ICASE
NASA Langley Research Center
Hampton, Virginia

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 2001

AN ANALYSIS MECHANISM FOR AUTOMATION IN TERMINAL AREA

STAVAN M. PARIKH*

Abstract. This paper describes an Air Traffic Simulator (ATS) that provides a simulation capability for terminal area automation. The ATS is capable of realistic terminal area modeling, automation performance and feasibility studies, and system wide fault-tolerance analysis. The ATS is evaluated through modeling of self-spacing and self-merging of aircraft following Standard Terminal Arrival (STARs). These algorithms are described in detail and preliminary analysis results are provided. The analysis exposes the limitations of the algorithms due to their integration. Possible modifications to overcome these limitations are suggested. Future work planned for the ATS is described. Overall the ATS is an easily extensible and powerful tool for preliminary analysis of new technologies.

Key words. self-spacing, merging, terminal area, RAPTOR, DAG-TM CE-11, modeling air traffic, simulation

Subject classification. Computer Science

1. Introduction. Automation is the key to improving the efficiency of the air traffic system. Before new automation technologies can be incorporated into the air traffic system, they need to be extensively tested and analyzed. This testing needs to be done not only for the feasibility and functionality of the technology itself, but also for its effect on the overall system in terms of performance and dependability. Due to the high cost and risk associated with actual testing it is not possible to test every new algorithm in an actual test environment. A practical alternative to an actual testing environment is provided by simulation. A successful technology at the simulation level can then be further tested in real-life conditions.

This paper describes a simulator designed to test various automation technologies in terminal area flight. It provides an in-depth look at the simulator so that it can be used for various other research problems. For evaluation of the simulator, the paper describes a self-spacing and merging algorithm for Standard Terminal Arrival (STARs). Finally, it presents preliminary simulation results of the self-spacing and merging algorithm and suggests possible improvements.

Simulation provides the means to see an algorithm in action, to visualize failures, and to study performance. Simulation also provides a global view of the system that enables the analysis of the algorithm's effect on the overall system. It allows for the study of fault tolerance of the system at a very low cost. This makes simulation an ideal test bed for new technology.

Development of new technology is needed throughout the whole spectrum of air traffic management. The Federal Aviation Administration (FAA) and relevant industries are pursuing this under the concept of Free Flight. One possible implementation of Free Flight, proposed by NASA, is the Distributed Air/Ground Traffic Management (DAG-TM) concept [1]. DAG-TM was formulated as a coherent set of solutions known as concept elements (CE) to a series of key Air Traffic Management (ATM) problems [1]. Some of these concept elements focus on terminal area. Terminal area is plagued with high traffic density and limited airspace making automation crucial for continuous movement of air traffic with minimal delays. In DAG-TM CE-11, self-spacing for merging and in-trail separation in terminal area are suggested as possible enhancements to reduce terminal area inefficiencies.

*9406 Tracey Lynn Circle, Glenn Allen, VA 23060 (email: stavan@virginia.edu). This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-2199.

Self-spacing is the concept where an airplane maintains required separation to its leading airplane with no assistance from ground control. Self-merging allows airplanes coming from different directions to merge onto a single path without any assistance from ground control. This reduces the workload of ground-control to that of sequencing aircraft and dealing with abnormal situations.

The rest of this paper is organized as follows: Section 2 provides an overview of structured air-routes in terminal area; Section 3 provides a description of the simulator; Section 4 covers self-spacing, self-merging, and sequencing algorithms; Section 5 presents preliminary results; Section 6 summarizes the goals of this research and discusses planned future work.

2. Structured Air-Routes. Traffic management in high-density airspace can be achieved by providing structured routes for the airplanes to fly on. For this purpose there are fixed arrival routes for each airport. These routes provide a series of waypoints that the aircraft follow to a terminal metering fix.

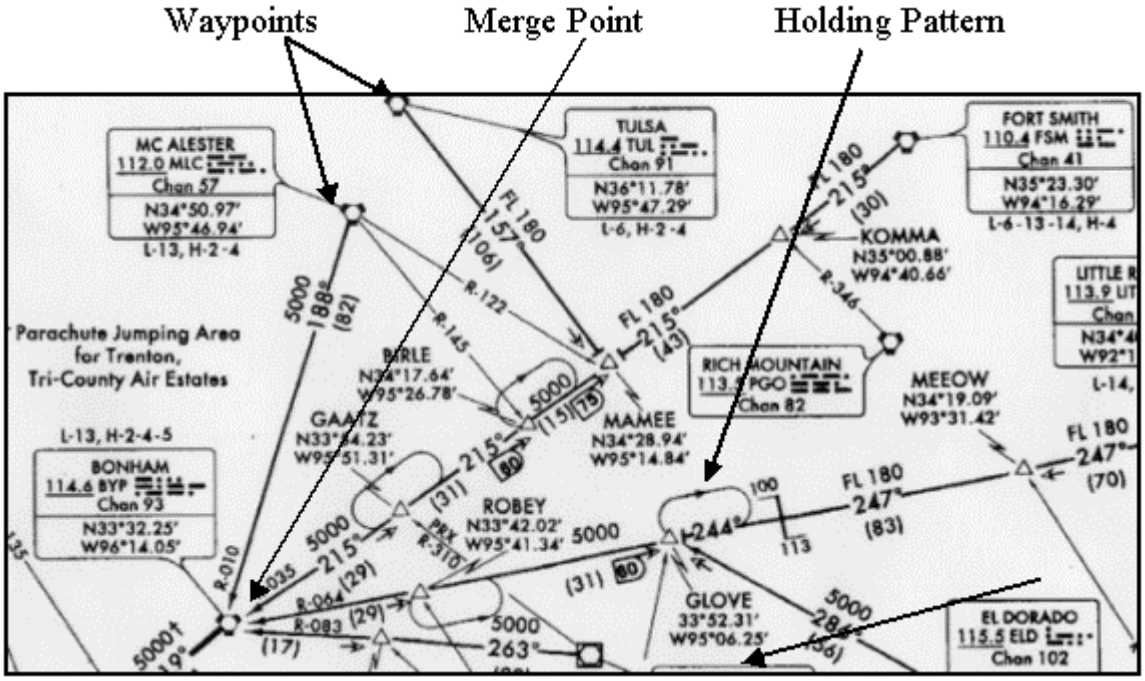

FIG. 1. *STAR Example*

As airplanes approach their destination airports from the enroute phase of flight they are vectored to predefined waypoints or metering fixes. Once on these waypoints, they fly predefined routes to enter the terminal area and then follow ground vectors to runway. The predefined routes for aircraft to enter terminal area are known as STARs (Standard Terminal Arrival). Fig. 1 shows an example of a STAR route. These routes cover a radius of approximately 200 nautical miles (NM) around the airports. Aircraft arrive at the waypoints and proceed along the indicated heading to the airport. The flight path defined by two waypoints is known as a flight segment. These waypoints are actually VOR (VHF Omni-directional Radio) beacons broadcasting their position to airplanes based on which, airplanes can determine their own position. Fig 1 shows that along with the name and position of a VOR a radio frequency is also indicated. Along with the waypoint and heading information, STARs also specify possible holding patterns that the aircraft might be requested to follow. The number in parenthesis below each segment provides the length of the segment in nautical miles. Also indicated are minimum altitude and recommended flight levels for each flight segment.

A good modeling system for terminal area needs to provide a way to simulate airplanes following routes, reasonable acceleration and deceleration of airplanes, and realistic traffic. This capability is provided by the Air Traffic Simulator (ATS).

In the rest of this paper whenever we consider two aircraft, the aircraft that is directly referred to will be known as ownship and its leading aircraft will be known as traffic. Also the nominal speed for each segment is the optimal speed predefined for that segment of flight.

3. The Air Traffic Simulator (ATS). The ability to study interactions between airplanes in a high-density realistic traffic environment is crucial for the development of new technologies.

3.1. Overview. The Air Traffic Simulator has been written around a core network simulator – RAPTOR developed at the University of Virginia [5]. In RAPTOR a network can be described through a topology and connections. RAPTOR also provides the ability to attach vulnerabilities to different parts of the network and the ability to induce symptoms in the model. Once these vulnerabilities are assigned to different parts of the system, faults can be induced into the network during simulation. The effects of these faults on the whole system can then be studied.

The ATS uses RAPTOR to build an air traffic network. Fig. 2 shows the structure of ATS. The Core of the simulator is built around RAPTOR. To build the network, RAPTOR takes a topology and connections description as input. The topology describes each node to be simulated in the network. For the ATS two special nodes are the

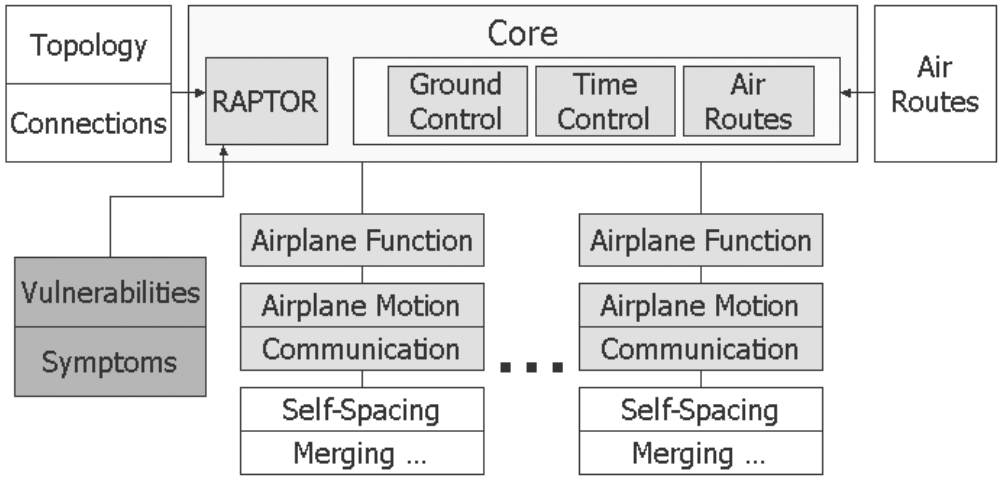

FIG. 2. *Air Traffic Simulator Structure*

ground control and timing control nodes. These nodes act as the ground controller and automatic traffic generator respectively. Another independent module within the ATS is the Air Routes module, which provides a way to describe any structured air-routes. This along with ground control, and timing control forms the other part of the core of the ATS. Also described within the topology is the number of airplanes required in the model. Each airplane runs an Airplane Function. These airplane functions may or may not be the same. When a new airplane is initiated into the airspace by timing control an air route is assigned to it and through the motion control in the airplane function the airplane is able to fly that route.

Another feature of the ATS is the ability to induce Vulnerabilities and Symptoms into the model [5]. Through this, known vulnerabilities of a network can be studied and a fault-tolerance analysis can be carried out on the model. Currently no work has been done in this area but is planned for the future.

The ATS uses an internal clock provided by RAPTOR. Through this the model can be time stepped to allow realistic modeling. This time is advanced when all airplanes are waiting on a time-step or waiting to receive a message from another airplane. Currently each model time-step is set to ten seconds of real time but this can be modified through the modification of a constant.

The total airplanes that can be simulated over time through the simulator is unbounded. The number of airplanes that can be simulated in parallel is bounded only by available computing power. Fig. 3 displays a screen shot of the ATS display. This display developed by Brian Rowe at the University of Virginia provides a visualization of the algorithms being tested by ATS making it easier to study algorithms and system wide fault-tolerance.

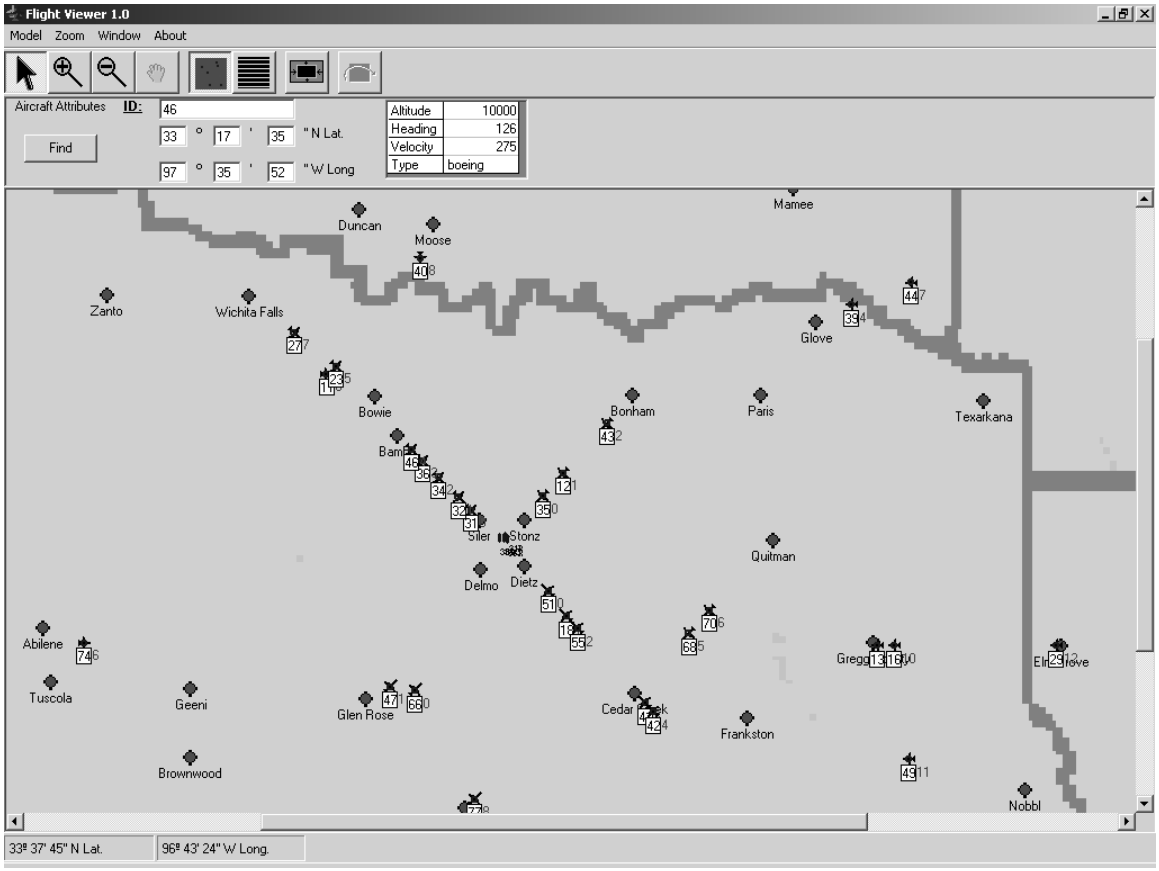

FIG. 3. *Air Traffic Simulator Display*

3.2. Ground Control Function. The ground control function simulates the function of central computer in contact with all aircraft in terminal area airspace. This function is responsible for airplane sequencing and keeping track of all airplanes in the airspace. It provides leading and trailing airplane information to all aircraft. The airplane sequences are updated as necessary when an airplane lands or merges out of sequence. Such a failure is also logged for future analysis. The sequencing numbers generated by ground control are for internal use of the algorithm. These numbers are not necessarily the same as the sequence numbers given to pilots by ATC. This is because unlike the ATC, when an airplane lands all following internal sequence numbers are decremented to keep computation simple.

3.3. Time Control Function. The Time Control is responsible for randomly inducing airplanes into the airspace at different times. It is also setup to reuse the airplane nodes of airplanes that have landed. This allows for continuous operation of the model over long periods of time with minimal resource requirement, that is the number of nodes defined in the topology. Because of the reuse of nodes, the number of airplanes in the topology is the total number of airplanes that can be in air at any given time and not the total number of airplanes that fly through the course of the simulation.

3.4. Air Routes. The function of the Air Route module is to provide an instance of air routes, such as STARs or TRACON, to be simulated for a given airport configuration. It takes as an input a structured Air Route file and uses it to make planes fly along fixed trajectories. Once a route is assigned to an airplane, the Air Route module then provides all the information necessary for the airplane to fly that route. Currently Air Routes do not incorporate holding patterns. An example input file for air-routes is described in section 4.5.2.

3.5. Airplane Function. This function consists of the spacing and merging algorithms. A simple modification to this code can allow for testing of many different algorithms thus making this simulator a very powerful tool for research.

The basic framework of this function sets up communication between ownship and traffic, and ownship and ground control. The data that is communicated is encapsulated in an air-message that can be modified to allow for different data exchange.

The airplane motion is governed by the speed advisories generated by the spacing and merging algorithm. It uses a very simple law of motion to compute distance and acceleration of airplane on every time step of the model. Currently the simulator uses a flat earth model with a constant assigned to the conversion from degrees to nautical miles. This proves adequate for now and later can be replaced with a more accurate model.

3.6. Simulation Inputs. The model takes as input three data files. Each of these is described here.

3.6.1. Topology File and Connections File. These files are formatted as prescribed by the RAPTOR model [5]. These files affect the maximum number of aircraft that can be simulated simultaneously in the airspace. For the purposes of the ATS a topology and connections file generator is provided that will generate the correct input files. This file generator takes the number of aircraft, and the topology and connection filenames as inputs.

3.6.2. Air Route Data File. This file accepts structured routes to a runway. This file can be modified to model different airspaces. The structure of the file is explained through an example. For this example refer to Fig 1. In this example we will create a text file for airplanes to enter at McAlester, Tulsa or FortSmith and then fly towards Bonham from where they fly on towards the airport. In the case of this STAR, Bonham is the final merge point.

The General structure of the route is as follows

Number Name < Waypoint Position> << Heading | Next Position > … < Heading | End position >>
Entry Point or Not

The first segment defines the final segment of flight starting at the final merge point towards the runway. In this case it is:

1 Bonham < N 33o 32.25' : W 96o 14.05' > << 219 | N 32o 58.37' : W 96o 54.17' >> N

In the case of the final segment there are two changes that are made to the general structure. The route number is prefixed with a '#' and after the end position a runway name is added. So In this case the first line actually is:

#1 Bonham < N 33o 32.25' : W 96o 14.05' > << 219 | N 32o 58.37' : W 96o 54.17' >> < RNW 17> N

Each flight segment that can lead to a waypoint on Bonham is then listed under with numbering proceeding as 1.x. In this case McAlester and FortSmith lead directly to Bonham. So they are listed as follows.

1.1 McAlester < N 34o 50.97' : W 95o 46.94' > << 188 | N 33o 32.25' : W 96o 14.05' >> Y

1.2 FortSmith < N 35o 23.30' : W 94o 16.29' > << 215 | N 34o 28.94' : w 95o 14.84' >
< 215 | N 33o 32.25' : W 96o 14.05' >> Y

All segments that can lead to any of the waypoints within these merge points are nested even more as 1.x.y. In this example Tulsa leads to a waypoint on FortSmith. So it is listed as:

1.2.1 Tulsa < N 36o 11.78' : W 95o 47.29' > << 157 | N 34o 28.94' : w 95o 14.84' >> Y

This nesting can be repeated if there are any segments that merge onto Tulsa and so on. When all the segments are listed the end of the star is marked by a '#'. Multiple stars can be added in the same file by repeating the above process with a different route number.

The above example will appear as follows in the air routes file:

#1 Bonham < N 33o 32.25' : W 96o 14.05' > << 219 | N 32o 58.37' : W 96o 54.17' >> < RNW 17> N
1.1 McAlester < N 34o 50.97' : W 95o 46.94' > << 188 | N 33o 32.25' : W 96o 14.05' >> Y

1.2 FortSmith < N 35o 23.30' : W 94o 16.29' > << 215 | N 34o 28.94' : w 95o 14.84' >
< 215 | N 33o 32.25' : W 96o 14.05' >> Y
1.2.1 Tulsa < N 36o 11.78' : W 95o 47.29' > << 157 | N 34o 28.94' : w 95o 14.84' >> Y
#

Things to note about the air routes file structure:

- all route numbers must be unique
- all '<' , '>', '<<" , and '>>' must be preceded and followed by a space.

4. Description of Algorithms Used for Evaluation. For the purposes of this simulation we used the following self-spacing and merging algorithms.

4.1. Self-Spacing. Self-spacing aims at maintaining minimum separation between ownship and traffic without assistance from the ATC. For this simulation we use the algorithm developed by Abbott [2]. This algorithm provides a speed control law for generating speed commands for the airplanes. The basis of the speed control law is for the ownship to follow its traffic such that it is at time $t+\Delta$ where traffic is at time t. Here Δ is the optimal time separation between the two aircrafts.

The speed control law provided is implemented in the simulation by the following equations [2,3]:

(3.1) $$\text{Speed} = \text{BaseSpeed} + \text{SpeedError}$$

(3.2) $$\text{Speed Error} = 0.01 \text{ X Gain1 X RangeError} - \text{Gain2 X DeltaAcceleration}$$

(3.3) $$\text{RangeError} = \text{TrafficPosition} - \text{BasePosition}$$

(3.4) $$\text{DeltaAcceleration} = \text{OwnshipAcceleration} - \text{TrafficAcceleration}$$

Equation (3.1) provides the necessary aircraft speed to achieve time separation Δ. Equation (3.2), (3.3), and (3.4) provide the values need to compute (3.1). In (3.1), BaseSpeed is either the nominal speed for the given route or the leading airplane speed if time separation between ownship and traffic is close to Δ. In (3.3), BasePosition is the traffic position at t-Δ or target position based on minimum separation distance. Gain1 is 2.5 /ft and Gain2 is 2.5 sec [2]. One restriction placed on the Speed Error is that it be within a given speed limit (3.5) [3]. This allows for reasonable acceleration and deceleration commands to the airplane.

(3.5)
```
if (Speed Error > Speed Limit)
        Speed Error = Speed Limit
if (Speed Error < –Speed Limit)
        Speed Error = –Speed Limit
```

Another restriction placed on the Speed of the airplane is that it has to be within a fixed range around the nominal speed of the segment (3.6). This allows for limited deviation from recommended (nominal) speed of the segment. Different ranges of speed restrictions around nominal speed were tested to analyze the algorithm.

(3.6)
```
if (Speed > Nominal Speed + Speed Deviation)
        Speed = Nominal Speed + Speed Deviation
if (Speed < Nominal Speed – Speed Deviation)
        Speed = Nominal Speed – Speed Deviation
```

A combination of the above equations provides the implementation of the speed control law.

4.2. Sequencing. Sequencing is the task of assigning consecutive sequence numbers to airplanes based on their predicted arrival sequence to a runway. This is done by ground control on First Come First Serve (FCFS) basis, as it is simple and easy to implement. Brinton has shown that the delay benefits of an optimal sequence versus a FCFS sequence in terminal area are minimal [4]. As airplanes enter the airspace they are assigned a sequence number based on their estimated time of arrival (ETA). This is calculated based on a nominal speed and distance to runway. Upon entry of a new airplane to the STAR area, aircraft that have a later ETA are given new sequence numbers.

4.3. Merging. On the STAR approach there are multiple waypoints on which merging takes place. The idea of self-merging is to provide an automatic speed advisory to all aircraft such that merging takes place safely. Due to limited research in merging algorithms a suitable algorithm for this simulation was unavailable. So for the purpose of this research a very simple merging algorithm was developed. This algorithm combines with the self-spacing algorithm to generate new speed advisories. This algorithm was purposefully kept simple to allow for rapid prototyping.

To understand the merging algorithm we need to revisit the spacing algorithm. In the spacing algorithm the ownship follows its traffic such that it is at time t+Δ where traffic is at time t. Here Δ is the required time spacing. The merging algorithm uses airplane sequence numbers to dynamically determine a new time spacing to be maintained between two airplanes. This new time separation is a multiple of Δ.

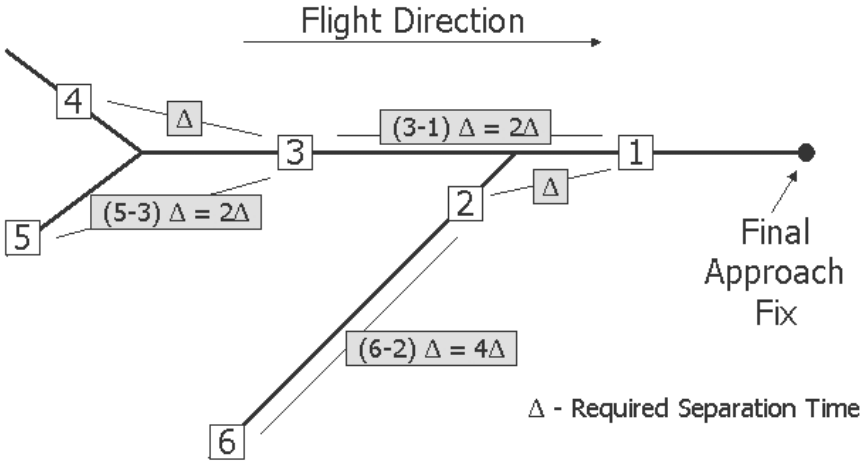

FIG. 4. *Merging Overview*

The algorithm is explained through the use of an example. Fig 4. shows six airplanes marked by blocks exposing their sequence numbers. We identify airplanes by their sequence numbers. Both airplanes 2 and 3 have 1 as their traffic. Airplane 3 is not aware of the speed and position of 2 as they are on different segments. Airplane 2 has traffic 1. The fact that 2 is the next number after 1 indicates that these airplanes will be landing consecutively. So they have to maintain separation Δ. Airplane 3 has traffic 1 as well, which indicates that airplane with sequence number 2 is going to arrive between them. So 3 should maintain separation 2Δ with airplane 1 in order to allow airplane 2 to come between them. Generally ownship has to maintain separation nΔ where n denotes the difference of sequence numbers between ownship and traffic (Equation 3.7).

(3.7) $\text{SeparationTime} = (\text{Traffic Sequence Number} - \text{Ownship Sequence Number}) \times \Delta$

With this new modification ownship now follows its traffic such that it is at time t+SeperationTime where traffic is at time t.

These algorithms are simulated in the Air Traffic Simulator (ATS).

5. Preliminary Results and Observations. With limited analysis the following has been observed.

- Self-spacing is able to maintain the required separation between traffic and ownship for flight along a continuous segment with no merges.
- Self-spacing combined with merging might fail. This failure might occur because the merging algorithm requires an arbitrary acceleration or deceleration of an airplane. As the spacing algorithm puts a speed bound on the airplanes, the airplanes are not able to meet the advised speeds for merging correctness and the spacing fails. In rectifying this situation by loosening or removing the spacing speed bounds the spacing algorithm may fail. This is due to the fact that the ownship accelerates to high speeds to get to the correct separation with the traffic. When this separation is achieved ownship starts to decelerate to match traffic speed but is unable to decelerate fast enough to maintain the spacing.

These observations show a need to either modify the algorithms to meet the spacing criteria or to pursue different algorithms. At this time a couple of possible enhancements have been proposed.

The first option is to incorporate holding patterns into the airplane sequencing. This will provide the airplanes the necessary time and distance to allow for proper spacing. The disadvantage here is the added flying time and the higher cost.

Another option is the use of dynamic merge points to space airplanes. The concept of dynamic merge points is explained by the following example.

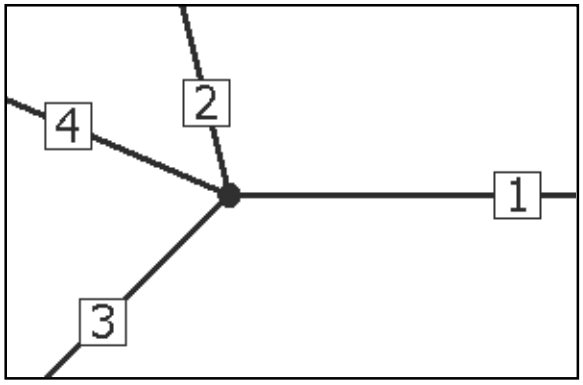

FIG. 5. *Spacing Failure at Merge*

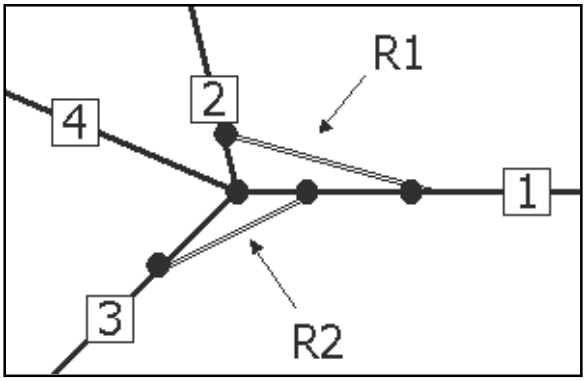

FIG. 6. *Dynamic Merge Points*

Fig. 5 shows a situation where airplanes 2, 3, and 4 are close enough that when they get to the merge point a spacing failure will occur. One option is to put 3 and 4 in a holding pattern to achieve the necessary spacing. The other option is to dynamically generate merge points as shown in Fig. 6. If new merge points as depicted in Fig. 6 by the end points of paths R1 and R2 are generated, then airplanes 2 and 3 can follow these paths respectively and the spacing can be maintained. This is because R1 and R2 are shorter then the original paths. The advantage of this scheme is that time is actually gained without loss of separation. Dynamic merge points can also be used to generate longer paths to avoid separation making it similar to holding patterns but can be of shorter duration than a holding pattern.

These are just preliminary observations and further work needs to be done to study all possible causes of failures and possible solutions to the same.

6. Summary and Future Work. This research has a two-fold goal. One is to develop a realistic and easily extensible simulator for researching high-density air traffic environment. The ATS aims at providing a tool to study various airborne automation technologies. Also it provides a platform for analyzing fault-tolerance in the system. The second goal is to analyze various airborne self-spacing and merging algorithms and to provide a viable algorithm. This algorithm can then be formally verified and tested in real conditions.

As described in the paper the first version of ATS is complete and preliminary analysis on the algorithms has been performed. Based on this analysis possible enhancements to the algorithms and simulators have been identified and need to be implemented. Further work in enhancing the simulator is planned.

At the same time research in other possible algorithms needs to be pursued. Furthermore the study of fault tolerance of the system needs to be undertaken once a viable solution for self-spacing and merging is found. For this specific faults critical to the overall system need to be documented. Systems vulnerable to these faults have to be identified and analyzed.

Acknowledgments. The author thanks The Formal Methods Group at ICASE and at NASA Langley and John Knight at the University of Virginia for providing motivation and continuous assistance throughout this research.

Also the author thanks Brian Rowe at the University of Virginia for providing the display for the simulator making the task of analysis much simpler.

REFERENCES

[1] *Advanced Air Transportation Technologies (AATT) Project*, Concept Definition for Distributed Air/Ground Traffic Management (DAG-TM), Version 1.0. NASA Ames Research Center – NASA Langley Research Center, 1999.

[2] T.S. ABBOTT, *Advanced Speed Control Law for Airborne, In-Trail Self-Separation*, Distributed Air/Ground Traffic Management Research Initiative. NASA Langley Research Center, 2001.

[3] T.S. ABBOTT, *Advanced Terminal–Area Approach Spacing*, Software Requirements Document, Version 1.0. Distributed Air/Ground Traffic Management Research Initiative. NASA Langley Research Center, 2000.

[4] J.E. ROBINSON, T.J. DAVIS, AND D.R. ISAACSON, *Fuzzy Reasoning Based Sequencing of Arrival Aircraft in The Terminal Area*, NASA Ames Research Center, 1997.

[5] *The Raptor Simulator.* Survivability Research Group, U. of Virginia, 10 June 2001. <http://www.cs.virginia.edu/~survive/raptor/>